Network Connectivity Proxy: Architecture, Implementation, and Performance Analysis

Bolla, R., Giribaldi, M., Khan, R., & Repetto, M. (2017). Network Connectivity Proxy: Architecture, Implementation, and Performance Analysis. *IEEE Systems Journal*, *11*(2), 588 - 599. https://doi.org/10.1109/JSYST.2015.2438639

Published in:
IEEE Systems Journal

Document Version:
Peer reviewed version

Network Connectivity Proxy: Architecture, Implementation, and Performance Analysis

Network Connectivity Proxy: Architecture, Implementation, and Performance Analysis

Raffaele Bolla, *Member, IEEE*, Maurizio Giribaldi, Rafiullah Khan, and Matteo Repetto

***Abstract*—Several studies in the last decade have pointed out that many devices, such as computers, are often left powered on even when idle, just to make them available and reachable on the network, leading to large energy waste. The concept of network connectivity proxy (NCP) has been proposed as an effective means to improve energy efficiency. It impersonates the presence of networked devices that are temporally unavailable, by carrying out background networking routines on their behalf. Hence, idle devices could be put into low-power states and save energy. Several architectural alternatives and the applicability of this concept to different protocols and applications have been investigated. However, there is no clear understanding of the limitations and issues of this approach in current networking scenarios. This paper extends the knowledge about the NCP by defining an extended set of tasks that the NCP can carry out, by introducing a suitable communication interface to control NCP operation, and by designing, implementing, and evaluating a functional prototype.**

***Index Terms*—Energy efficiency, green networking, home gateway, network connectivity proxy (NCP), Universal Plug and Play (UPnP).**

I. Introduction

SEVERAL studies about usage patterns have shown that computers are only used for a small fraction of the time that they are switched on; often, they are powered even if nobody is attending to them [1], [2]. The main reason behind this behavior is to leave computers present and available on the network, in order to carry out background network activity (responding to discovery protocols such as NetBIOS and Universal Plug and Play, renewing soft states such as for Dynamic Host Configuration Protocol, confirming "online" status for Voice-over-IP and chat applications, etc.). The worst implication of this fact is that they waste a lot of energy, which sums up to several TWh worldwide [2], [3].

Computers can be in different working states, which correspond to different power consumption and different operational latency. Table I shows a simple abstraction of the main working states[1] [5]. A computer is *active* when it is performing some tasks (e.g., CPU processing, seeking data from storage, memory, or cache). When *idle*, the device is not performing any useful operation, but the hardware is fully functional and ready to start. In *standby*, part of the hardware is powered off, and resuming it to full operation will take a little time (usually of the order of a few seconds). Finally, *off* means that the device was shut down and bringing it back to the active mode will need a complete boot sequence (taking several seconds).

TABLE I
Typical Working States for Computers. Indicative Power Consumption Was Taken From the ENERGY STAR Program

Common name	ACPI state	Description	Power
Active	S0	All hardware fully powered on.	> 40W
Idle	S0	All hardware fully powered on; no operation carried out.	> 30W
Standby	S3	CPU is powered off. RAM is powered on.	< 5W
Off	S5/G3	All hardware is powered off. Network cards may be on for remote wake up.	< 2W

Table I also includes indicative power consumption for each state (real values change significantly with the hardware type: desktop, laptop, workstation, server). Clearly, idle is a very ineffective state, because much energy is drawn to do nothing. Standby is specifically designed for reducing the energy consumption when the device has nothing to do and for quickly resuming to full operation. Standby states are already available in most computers; however, these features are seldom used because computers lose their network presence in these states.

The concept of network connectivity proxy (NCP) has been proposed for impersonating network presence of "sleeping" devices [6]. Impersonating network presence means that the NCP should carry on various networking tasks on behalf of other devices, so that they appear as fully functional and operational to other hosts [3], [7], and it should wake them up when their involvement is unavoidable. Hence, the NCP enables idle devices to enter standby, without losing their network connectivity, and that reduces the energy wasted by idle states.

Generally, NCP goals include the following: 1) preserving host reachability, e.g., responding to Address Resolution Protocol (ARP) requests and NetBIOS name queries, and sending DHCP lease renewals; 2) preserving host manageability, i.e., answering control and management protocols such as the Internet Control Message Protocol (ICMP) and the Simple Network

Manuscript received November 14, 2014; revised April 8, 2015; accepted May 23, 2015.

R. Bolla and R. Khan are with the Department of Electrical, Electronic and Telecommunications Engineering and Naval Architecture (DITEN), University of Genoa, 16145 Genoa, Italy (e-mail: raffaele.bolla@unige.it; rafiullah.khan@unige.it).

M. Giribaldi is with Infocom Srl, 16128 Genoa, Italy (e-mail: maurizio.giribaldi@infocomgenova.it).

M. Repetto is with the National Inter-University Consortium for Telecommunications (CNIT), 16145 Genoa, Italy (e-mail: matteo.repetto@cnit.it).

[1] In practice, computers are usually compliant with the Advanced Configuration and Power Interface (ACPI) specification [4], which entails more power states.

Management Protocol (SNMP); 3) maintaining application reachability, i.e., allowing new connections to be established; and 4) preserving application state, i.e., maintaining transport layer connections alive and sending/responding to application and protocol specific periodic heartbeat messages.

Until now, most work has focused on responding to stateless protocols (ARP, ICMP, NetBIOS) and to move entire applications or part of them between the NCP and its clients. Nobody has still tackled how the NCP interacts with its clients, for the purpose of registering what kind of behavior is requested and of notifying when to start/stop the operation.

In this paper, we propose a framework that includes a set of basic and general routines that the NCP carries out on behalf of its clients and a suitable communication interface [8]–[10]. We also describe the current implementation of the framework, which runs both on stand-alone devices and network equipment (home gateway) and which also deals with practical matters such as the presence of private addresses, i.e., it manages *Network Address Translation*. The software is flexible enough to exploit different protocols for the communication interface; we have already integrated UPnP and have defined a specific service template for the NCP. This paper enhances our previous work on the same matter by extending the set of aforementioned routines and by presenting the relevant outcomes we got from the functional and performance evaluation of the framework.

The rest of this paper is organized as follows. Section II reviews related work. Section III briefly describes the main components of the NCP framework. Section IV gives an overview of the current implementation. Section V lists some of the most relevant issues that arose during the implementation and evaluation of the software; it also reports performance evaluation of the NCP software. Finally, Section VI gives our conclusions and plans for future work.

II. Related Work

The topic of managing connectivity on behalf of sleeping devices has been covered by researchers with a focus on different aspects.

Most implementations have focused on the Transmission Control Protocol (TCP) and management protocols such as ARP, ICMP, and the Internet Group Management Protocol (IGMP): they usually answer ICMP echo-request messages and ARP requests and wake hosts up on incoming TCP SYN segments or NetBIOS name queries [1], [2]. The first attempt to manage on-going TCP sessions was limited to inhibit the remote peer from sending any data during sleep periods, by advertising a "zero-window" condition. The addition of a new option in the TCP header to advertise the next power state (standby or active) was also envisioned [11], as well as the use of a "shim" layer between applications and the legacy socket interface, which sets up and closes TCP connections at each power transition and hides this behavior to applications [3]; both of these approaches require the remote peer to be aware of the power state of the device. The use of an external SOCKS proxy was envisioned for splitting TCP connections at the proxy and for keeping the peer unaware of power state transitions [6]. However, this forces the proxy to relay data between the two connections when the device is active, posing scalability and performance constraints.

There have also been a few proposals for proxying both high-level protocols such as UPnP [12] and specific applications as Gnutella [13] and Jabber [14]. A further step was done in Somniloquy, where "stubs" for specific applications can be added to the base framework [15]. An evolution of this approach consists in running virtual machines that load the images of sleeping devices and impersonate them during the sleeping periods [16]; this allows virtually supporting any application, without requiring specific implementations. However, it requires large processing power and memory to the NCP, and this solution is unlikely to scale for many clients.

Another interesting matter concerns architectural design. NCP functions can be deployed in smart Network Interface Cards (NICs), in network equipment (such as switches, access points, home gateways, and routers), and in stand-alone devices, with different considerations about additional power consumption and processing capabilities [17]. Many implementations targeted smart NICs because they are directly connected to the host (hence simplifying configuration of the NCP), draw very low current, and are traversed by all traffic intended to the host [2], [13], [15]. However, onboard processors have limited memory and processing capability. Placing the NCP on network equipment or stand-alone devices allows sharing its consumption and processing capability among several clients, with additional benefits in terms of energy savings [1], [6], [7]. In this case, more issues arise about the communication interface with the NCP and the interaction with current networking protocols.

Finally, the question about energy savings achievable by implementing the NCP concept has been also tackled [1], [2]. Traffic and use patterns of devices in different environments (office, home, dormitory) were analyzed; furthermore, an estimation about energy saving was carried out by considering four different types of NCPs, with different capabilities in terms of applications and/or protocols that could be managed.

Currently, no in-depth analysis has ever been conducted about the impact that proxying may have on networking operation and performance constraints that the NCP may face. Furthermore, nobody has considered the need and implications for a communication interface to control NCP operation.

III. NCP Architecture

The design of an NCP entails the presence of several components. Here, we provide an insight into NCP operation and point out the main architectural components that are needed to build the whole framework. We also analyze in detail the most relevant of them.

A. Overview of NCP Operation

The role of the NCP and the interaction with its clients are shown in Fig. 1. Devices on a subnet know an NCP service is present either by manual configuration or by a discovery protocol. They instruct the NCP about the tasks it should handle on their behalf (1). This operation includes the indication of

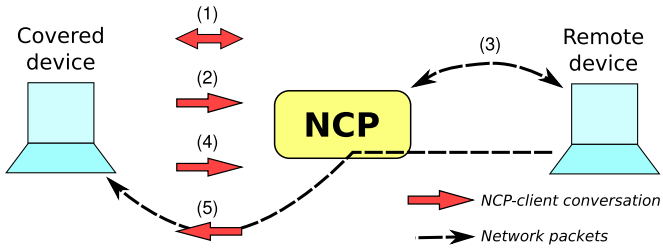


Fig. 1. Overview of NCP operation.

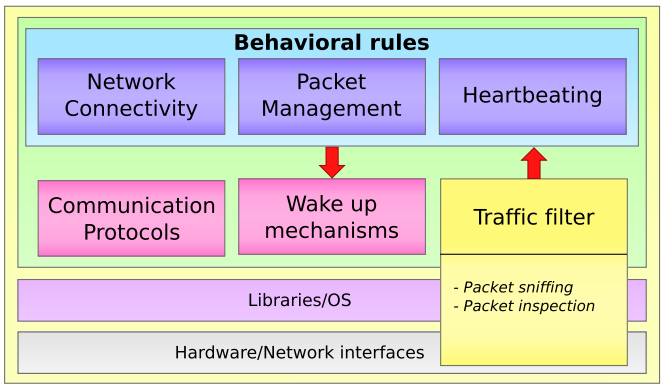


Fig. 2. NCP conceptual architecture.

when the tasks should be run (on the reception of specific packets or periodically), the action to be carried out (e.g., waking up the host, building and sending a packet), and any data that could be necessary to implement the tasks (IP addresses, port numbers, timeout values, etc.).

As soon as the internal policies decide to switch to the low-power mode, a device notifies the NCP that it must start to cover for it (2). From now on, the NCP catches all packets addressed to that client; this implies that network traffic intended to such device must be diverted toward the NCP by suitable network hooks (traffic diversion). The NCP carries out periodic tasks and handles incoming packets (3) according to the behavior previously settled. It keeps on covering for it until either the host autonomously wakes up (4) or a packet is received that needs specific processing by the host (5); in the last case, the NCP is in charge of waking up the host and forwarding the packet.

Every time a device wakes up, it notifies the NCP to stop covering for it. Traffic diversion is canceled, and the NCP forwards any buffered packet.

B. NCP Architecture

According to the operation described in Section III-A, the design of an NCP service entails several matters: the abstraction and representation of the tasks it can carry out, the communication protocol between the NCP and its clients, and the diversion of network packets intended to sleeping devices.

Fig. 2 shows a simplified view of the NCP framework. It consists of four main components: 1) behavioral rules, 2) traffic filter, 3) wake-up mechanisms, and 4) communication protocols.

Behavioral rules are the abstractions of the background routines that the NCP is able to carry out. They are roughly classified as Network Connectivity (responding to network protocols such as ARP), Packet Management (handling network packets), and Heartbeating (sending periodic messages on behalf of applications). Packet management encompasses several operations.

1) Buffering packets: The NCP buffers packets to avoid any loss or to delay client wake up.
2) Sending predefined packets: A packet that is known in advance and does not depend on any variable parameter is sent.
3) Building packets: The NCP understands the specific protocol/application and builds the response packets on its own.
4) Building packets by template: The NCP builds packets using predefined templates. The templates are filled in according to the instructions contained in the rule.

The execution of background routines also involves waking sleeping devices up, if they are requested to directly handle incoming packets. Wake-up mechanisms specify the methods to resume sleeping devices to full operation; currently, the de facto standard technology is Magic Packet, also known as Wake-on-LAN (WoL) [18].

Traffic filtering is the process of "sniffing" and inspecting packets addressed to sleeping devices, in order to trigger the execution of background routines. The traffic filter must only detect packets that are relevant for the set of registered rules and must discard any other.

Finally, communication protocols are meant to control NCP operation; they are used by client devices to register and to withdraw behavioral rules. In addition, communication protocols may also provide additional features such as automatic discovery and soft states. There might be more than one communication protocol present, in order to support different kinds of devices (e.g., standard PCs, sensors, and industrial equipment).

C. Behavioral Rules

The set of behavioral rules defined so far provides specific operations for basic connectivity protocols and generic tasks. The rest of this section briefly reviews them.

1) ARP Rule: The ARP rule answers ARP requests. The NCP provides its own MAC address in the response, in order to catch all packets intended to sleeping devices (we refer to this operation as "traffic diversion").

2) PING Rule: The PING rule answers ICMP echo requests. This is a diagnostic tool used by several applications (e.g., *traceroute*). The PING rule is only a simple example; further rules should be defined for other ICMP operations.

3) DHCP Rule: The DHCP rule periodically renews IP address leases with the DHCP server.

4) Wake-on-Connection Rule: The Wake-on-Connection (WoC) rule wakes devices up when a packet addressed to a given transport port is seen. This rule works both for connection-oriented (i.e., TCP) and for connectionless (i.e., UDP) protocols.

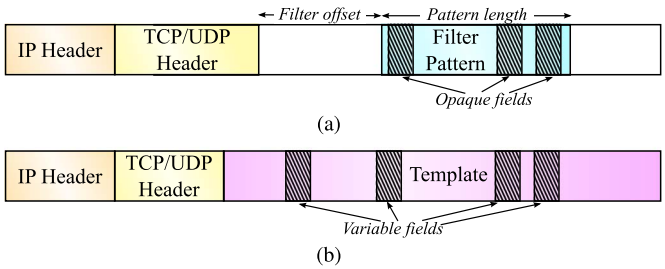


Fig. 3. Packet handling for heartbeating. (a) Pattern matching ignoring opaque fields. (b) Dynamic message generation by bit template and variable fields.

5) TCP Keep-Alive Rule: The TCP Keep-Alive (TCP-KA) rule maintains a given TCP session active by answering periodic TCP keep-alive messages. This rule also manages the reception of new data on the session, either by waking the client up immediately or after a given period or by advertising a zero-window condition to prevent the peer from sending data. This rule is not conceived to provide full TCP support for sleeping hosts: in particular, it cannot send/acknowledge application data.

6) HeartBeating Rule: The HeartBeating (HrtBt) rule provides a general framework to generate heartbeat messages for any application. This rule allows generating both solicited and unsolicited heartbeats: solicited heartbeats are triggered by incoming messages, whereas unsolicited heartbeats are sent periodically at predefined time intervals. For solicited heartbeating, incoming messages are compared against a given filter pattern. Fig. 3(a) shows how the pattern is sought in incoming messages, starting from the beginning of the application data, with the possibility of ignoring some "opaque" fields that change in every packet. The generation of heartbeat messages follows a template-based approach, which allows the clients to specify variable fields that should be computed and inserted dynamically by the NCP [see Fig. 3(b)]. The definition of variable fields includes the following: 1) position of the field within the template, i.e., the offset from the beginning of data; 2) length of the field; 3) type of the field, which establishes how its data are computed for each packet; and 4) data, specific for the field type. Currently, our NCP framework supports the following field types:

1) *Counter:* The datum is a sequence number; an initial value is incremented by a given step for each packet.
2) *Timestamp:* The datum is computed by adding the elapsed time to the reference clock provided by the client at registration.
3) *RandomNumber:* The datum is a random number.

D. Network Traffic Diversion

Traffic diversion redirects packets intended to sleeping devices toward the NCP. Network traffic diversion is always required, but the case the NCP is deployed on board the device's NIC.

In LANs, traffic diversion simply implies binding the NCP's MAC address with the IP address of the sleeping clients. This objective is achieved by answering ARP requests on behalf of sleeping clients (as described by the ARP rule) and by sending unsolicited "gratuitous ARP" packets, in order to update ARP caches on the other devices. Obviously, ARP caches must be updated again to the original MAC when the covered device wakes up. This solution is compliant with RFC 826 [19] and RFC 5227 [20].

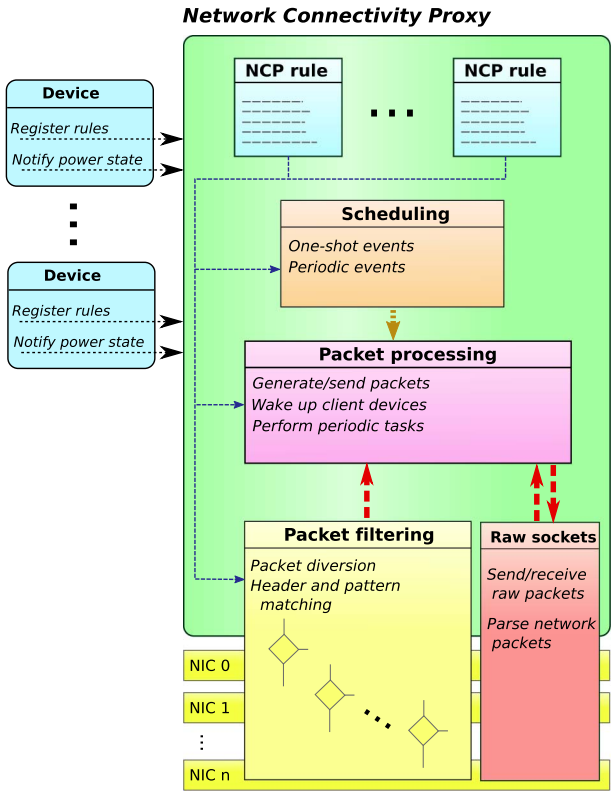


Fig. 4. Software architecture for the NCP agent.

IV. Implementation

The implementation of the NCP framework encompasses two software agents, one for the NCP service and one for clients. Both agents interact by the communication protocol, as outlined in the description of the entire framework (see Section III-A).

A. NCP Agent

The software architecture for the NCP agent is shown in Fig. 4. It includes the database of behavioral rules, packet processing, scheduling, packet filtering, and raw sockets. In addition to that, there is the communication interface to receive commands from clients (registration of their rules and notification of their power state transitions) and the main NCP logic that orchestrates all the components.

Each rule contains the description of one specific task for the NCP. Rules are made of two parts: the condition and the action. The former specifies the event that triggers the execution of the latter. Conditions consist of either matching criteria on packet content or time intervals. Actions implement the routines that

the client has delegated to the NCP (e.g., ignore the packet, buffer the packet, generate/respond to the packet, or wake up the sleeping device). The NCP maintains a database of behavioral rules, which are dynamically registered and withdrawn by its clients. Rules are inactive as long as the corresponding client is awake and are activated once it enters into low-power mode. Activation of behavioral rules for a client implies three tasks: 1) traffic addressed to the client is diverted toward the NCP; 2) the filtering engine is set to match traffic patterns contained in those rules (e.g., source/destination IP addresses and port numbers and protocol); 3) any periodic operation is scheduled at the appropriate time; and 4) any preliminary operation is carried out, e.g., to infer parameters as remote servers (DHCP), sequence numbers (TCP), and so on.

Packet filtering inspects packets and catches those that match the conditions in active rules. The inspection process considers header information (source and destination addresses, protocol, source and destination ports, protocol-specific flags and options) and packet content (bit patterns, application-specific headers and data).

Scheduling triggers routines as the time elapses. Triggering is one shot (i.e., only one trigger occurs) or periodic (i.e., many triggers are generated).

Packet processing performs the action entailed by each NCP rule, when triggered by the relative condition.

Finally, raw socket is the application programming interface (API) that enables to read and write network packets directly at the lowest layer, thus bypassing the entire built-in networking stack. Raw sockets are mostly needed to build network packets on behalf of client devices (i.e., with a different IP address); this operation would otherwise be prohibited by the operating system (OS).

The orchestration logic is responsible for updating the database of behavioral rules and for activating/deactivating them every time a device enters/exits standby, respectively.

The software is entirely written in C++ and designed for Linux; thus, it runs on off-the-shelf computers and on many embedded devices. Packet filtering uses the Pcap library.[2] Such library provides a common interface to the specific underlying packet capture facility implemented by the OS. It works in user space and filters packets by optimized Berkeley Packet Filters, which are derived from a text string in human-readable format.

Our implementation also supports migration of TCP sessions [21], which is used for the Heartbeating rule. Clients push the state of their TCP connections to the NCP just before entering standby, and the NCP keeps on sending/receiving messages on their behalf on the same connection in a totally transparent and seamless manner.

In addition, when the agent runs on a box at the periphery of a private domain (e.g., an home/access gateway), the software manages network address translation. It retrieves information about the current IP/port translation mappings and applies them to raw packets sent and received on behalf of client devices on the "public" interface.

B. Client Agent

Client software is basically an agent conceived to provide the applications with a common interface to interact with the NCP. It offers a simple API to register and to withdraw behavioral rules, freeing each application from the burden of managing the details of the communication protocol and the abstraction of rules. The agent automatically detects when the device is going to standby and when it has resumed, and notifies the NCP accordingly. Different mechanisms are used for this purpose, depending on the OS. The client software also manages automatic discovery of the NCP service in the network.

We built client software for both Linux and Windows; the Linux version includes an API for TCP migration [21].

C. Communication Protocol

The communication protocol in the current implementation is based on UPnP; the UPnP framework and operation are described by the UPnP architecture [22]. Basically, it distinguishes between controlled devices (CDs) and control points (CPs). CDs play the role of servers, responding to requests from the CPs. The UPnP description for a CD describes the characteristics of the device and its capabilities. The description is made of two logical parts: device description, which describes physical or logical containers, and service description, which includes the list of commands (actions) the service responds to, parameters (arguments) for each action, and variables that can be subscribed by the CPs.

To define a standard UPnP communication interface for the NCP, we have formalized a reference template for the "NCP service" and have wrapped it into an Internet gateway device, which is already standardized by the UPnP Forum.[3] We have introduced a service type of *NetworkConnectionProxy:1* and a service identifier of *NetworkConnectionProxy1* for the NCP; we have defined the full list of actions (name, arguments, types of data, allowed values, and default values) needed to implement the communication interface. Further details about the usage of UPnP in the NCP framework are available in previous papers [8].

V. Evaluation of the NCP Framework

Effective and seamless usage of the NCP faces a number of issues that are difficult to take into account and to evaluate during the design stage. Our implementation was indeed conceived to find an answer to the many questions past studies on this topic still leave open, particularly the ones concerning performance matters and the estimation of the amount of energy that could be saved in different scenarios.

Our analysis was conducted both in qualitative and quantitative terms. We investigated practical concerns that arise when the NCP concept is applied to legacy networks. We also carried out detailed measurements to point out technical and performance constraints of the selected architecture and software implementation, particularly when it runs on low-power

[2]Tcpdump & Libpcap. Web site: http://www.tcpdump.org.

[3]Internet Gateway Device (IGD) V 1.0. URL: http://upnp.org/specs/gw/igd1/.

devices. Finally, we put together our findings to estimate how much energy could be saved in home and office environments.

A. Impact of the NCP on Legacy Network Operation

A number of practical issues were noticed during the development and validation phases that had never been considered in previous studies. Most of them came from the fact that the NCP runs in parallel to the main networking stack of the OS, but it is not a part of it. We report a brief list of them, together with discussion about possible countermeasures to solve them or to mitigate their effects.

1) Host Unreachable Errors: Traffic diversion is conceived to alter the path followed by packets on their way to the destination, but does not end them automatically at the NCP. Hence, packets continue their trip once they have crossed the NCP and inevitably trigger ICMP host unreachable errors. Therefore, the NCP must take care of dropping packets intended to sleeping clients. This can be easily accomplished by Netfilter.[4]

2) Spoofing of Caches in Network Switching Equipment: The NCP sends ARP packets for traffic diversion. Sending exactly the same packet that would be sent by the sleeping device implies using its MAC address as the source of the packet; however, this spoofs the learning database of switching equipment, which now believes that the device is on the same port as the NCP. This way, when that device wakes up, its packets will be switched on the wrong port, thus delaying its restoring full network connectivity. Hence, the MAC address of covered devices must never be used as source address of network packets the NCP sends. This trick works well, as the resolution process only considers the information carried by the ARP packet and safely ignores the datalink header.

3) Conflicts in Transient Periods: The transition to and from the sleeping state requires some time. The NCP is notified of the transition slightly before the host goes to sleep and slightly after it resumes its operation, but it cannot exactly know when the device stops and when it starts again to use the network. This means there is a short transient where operation of both the NCP and the covered host might potentially conflict. However, since the NCP only intervenes when the host is idle, the likelihood of both getting the same packets is almost negligible; indeed, we did not notice any such problem in our evaluation.

4) Sequence of Operations: There are several operations that the NCP must undertake to start and to stop covering a device: diverting traffic, starting packet processing and dropping, and sending buffered packets. The order of these operations should be carefully selected to avoid side effects such as lost packets and host unreachability errors. For instance, if packets are diverted and reach the NCP before it has activated the corresponding rules, they are processed by the OS networking stack, and they trigger one of the undesired behaviors listed earlier.

5) Answering Local Applications: The operation of the NCP requires sending raw packets outside of the standard networking stack. Raw sockets are explicitly designed to send packets out of the box: they deliver packets created by the user to the NIC and do not check the destination. Packets addressed to the same box are transmitted on the medium but are not received locally. This means that applications running on the same host as the NCP cannot receive answers from the NCP. The current implementation provides a "raw loopback" function based on the Linux tun/tap driver, which sends back packets to the same host they are generated from.

[4]The netfilter.org project. URL: http://www.netfilter.org/.

TABLE II
HARDWARE PLATFORMS USED FOR EVALUATION OF THE NCP

Vendor and model	CPU	Memory	Power [W] (Idle ÷ Max)
Raspberry Pi	ARMv6	512 MB	3.6 ÷ 3.8
Jetway JNC9C-550-LF	Intel Atom N550	2 GB	24.8 ÷ 27.3
Lantiq EASY 80920	ARM VR9	64 MB	6 ÷ 6.4

6) Local Address Resolution: ARP cache entries on the device hosting the NCP are not updated by responses sent by the same NCP. Even with the raw loopback workaround (see Section V-A5), the kernel still does not receive ARP responses. To overcome this issue, the NCP must add static cache entries and set them back to dynamic when such clients get awake.

7) Early Wake-Up Detection: When a covered device wakes up, it must notify the NCP. If the notification is carried by UDP, there are no particular problems. If TCP is used, the three-way handshake should take place, but no packet reaches the client because the NCP is still covering that host. As a result, the TCP connection fails. A simple workaround is "early wake-up detection," which infers the current status of a covered host by its network activity. When a packet is seen that comes from a sleeping device, the latter is inferred as "awake." Early detection checks for the source MAC address, to avoid getting confused by IP packets sent on behalf of the covered host and maybe reflected by some other device.

8) Duplication of Buffered Packets: To improve performance, the NCP buffers packets that have triggered host wake up. As soon as the NCP receives the wake-up notification, it sends such packets to the covered device. These packets may be caught by the NCP filtering engine and queued again; in this case, every time the host wakes up, all previous packets are captured, and the queue grows indefinitely. The solution is rather simple: disable packet processing before sending out buffered packets.

B. Performance Analysis

Performance analysis was carried out to assess how many devices and behavioral rules our architecture can support. Note that the implementation mainly targets small sites (e.g., homes and small offices), where the number of client devices is usually limited (laptops, desktops, smartphones, network printers, etc.). Nevertheless, scalability is an important issue to maximize energy savings, since the NCP is expected to run on very low power devices, which are constrained in terms of available processing power and memory. As a matter of fact, hardware of such kind should support a reasonable number of devices simultaneously, each of which can register several behavioral rules.

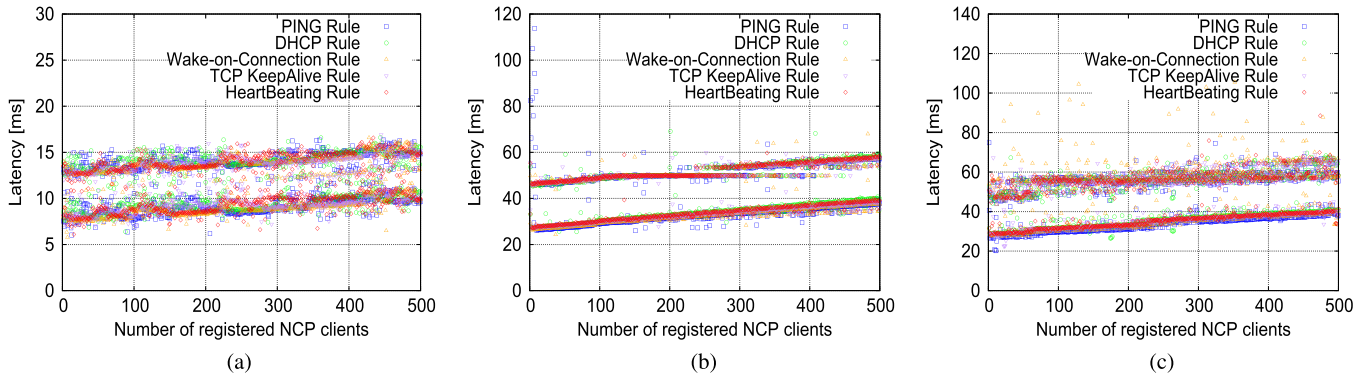


Fig. 5. Latency in processing registration requests for different behavioral rules on different hardware. The lower bands of values come from NCP processing, whereas the higher bands also include UPnP latency. (a) Jetway. (b) Lantiq. (c) Raspberry.

Our analysis considered three different platforms that are suitable to run the NCP software: one compact PC with a low-power CPU (Jetway), a home gateway (Lantiq), and a credit-card-sized computer (Raspberry). Table II provides a summary of the main characteristics of these devices, both in terms of processing power and energy consumption. These devices are interesting targets for NCP deployment: they represent network equipment (Lantiq) and stand-alone devices, even with negligible power consumption (Raspberry).

Our analysis involved several aspects, raising concerns about latency of operation, processing power, and memory requirements. In particular, we took into account the registration of behavioral rules (see Section V-B1), CPU and memory requirements to store and to process the rules (see Section V-B2), activation of behavioral rules (see Section V-B3), the execution of routines on behalf of sleeping clients (see Section V-B4), the overhead of the communication protocol (see Section V-B5), and the device transition to/from standby (see Section V-B6).

Performance evaluation was conducted in a synthetic scenario, where fake clients registered a set of behavioral rules with the NCP and notified power state transitions. This way, we could emulate the presence of different devices, each with its own networking parameters (IP and MAC addresses) and opened connections, even if they were not really set up. We could not carry out our trials with real clients because of the large number of devices considered, which makes unfeasible the setup of all the simulated connections. NCP performance is not affected by the synthetic scenario; indeed, we will see that, for certain parameters, the synthetic scenario is a worst case emulation.

1) Registration of Actions by Clients: We sequentially registered a set of rules (1 Ping, 1 DHCP, 1 WoC, 1 TCP-KA, 1 HrtBt) with the NCP for an increasing number of devices and measured the time taken for each registration. Fig. 5 shows both the NCP processing time and the total time seen by the clients (the latter includes the delay introduced by the UPnP protocol). The registration procedure only takes few dozens of milliseconds, independently of the rule type; the dependence on the number of registered client devices is quite limited. Clearly, latencies depend on the processing capability of the platform: as a matter of fact, the Jetway introduces much lower delay than the Lantiq home gateway and the Raspberry Pi. Small latencies mean that the NCP software can deal with multiple concurrent registrations.

2) CPU and Memory Requirements: Target platforms under consideration are equipped with low-power processors; some of them have a few hundred megabytes of memory only. Hence, the NCP software should run with minimal CPU and memory requirements.

To assess memory usage, we registered an increasing number of devices with the NCP. We collected the amount of memory assigned to the NCP process after each device had registered its set of rules. We considered four different scenarios: basic *Network Presence* (1 PING rule) for dynamically or statically configured hosts (namely, with/without 1 DHCP rule, respectively), *Reachability* (1 PING, 1 DHCP, and 1 WoC rule), and *Each-Rule-Once* (1 PING, 1 DHCP, 1 WoC, 1 TCP-KA, and 1 HrtBt rules).

Fig. 6 shows the memory usage versus increasing number of client devices. Our experiment was done under two conditions: all devices remain active and every device switches to standby after the registration. In the second case, we did not generate traffic toward the sleeping devices, in order to provide a baseline assessment. As expected, the memory usage depends on the number of rules requested by each client. While the memory usage increases almost linearly when devices stay awake, the rise is a bit faster when they sleep, due to the need to catch and inspect packets. The Lantiq home gateway was able to register up to 120 client devices (for Each-Rule-Once), beyond which the latency became too large. Apart from this limitation, the differences in the amount of memory consumed on different hardware are negligible. Furthermore, we observed that CPU usage was very low during registration of rules.

The CPU is mostly used when the NCP is covering for devices. We registered a PING rule for a device, and we put it into standby. Then, an increasing number of ICMP packets intended to the sleeping client was generated, until the CPU of the platform hosting the NCP reached 100% utilization. To better understand the impact of NCP operation on the CPU usage, we carried out another experiment. We injected the same ICMP traffic load in the same hardware platforms, without running the NCP.

Fig. 7 shows how CPU usage rises with the traffic load; the difference between the two experiments changes substantially

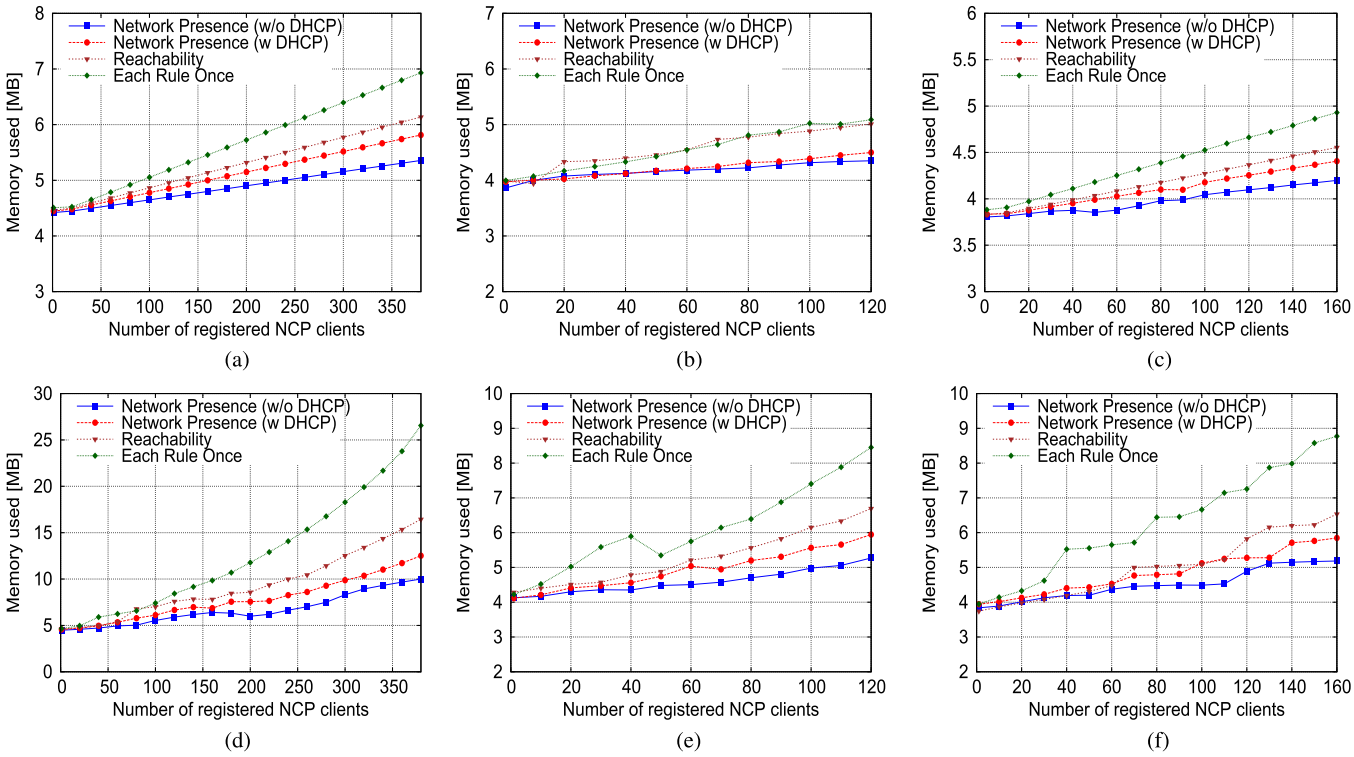


Fig. 6. Memory used by the NCP software on the different hardware platforms versus increasing number of registered client devices. Memory was measured both when all covered devices were active (NCP had nothing to do) and when all covered devices were in standby (NCP was filtering packets). (a) Jetway: awake clients. (b) Lantiq: awake clients. (c) Raspberry: awake clients. (d) Jetway: sleeping clients. (e) Lantiq: sleeping clients. (f) Raspberry: sleeping clients.

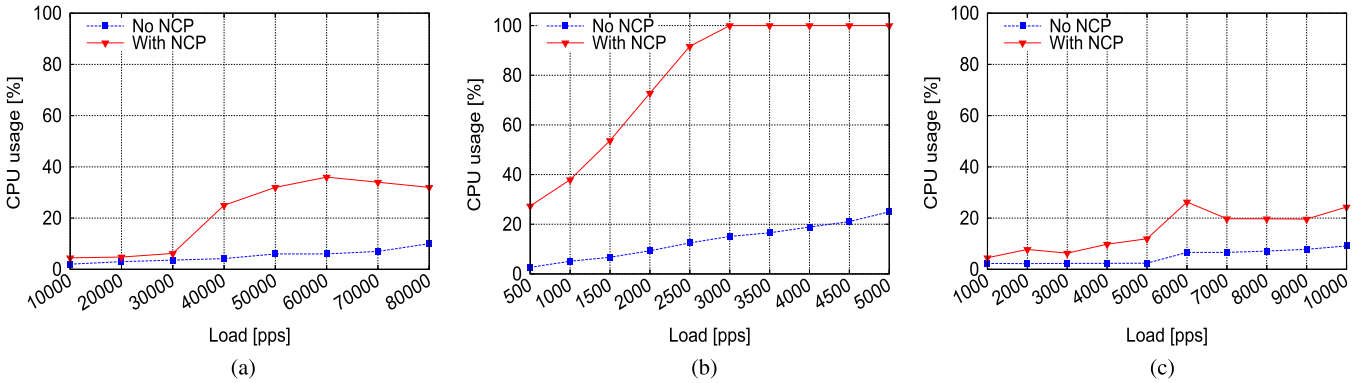


Fig. 7. CPU usage with increasing traffic load on different hardware platforms. Dashed lines (*No NCP*) indicate the CPU usage when the traffic is received but the NCP service is not running. Solid lines (*With NCP*) indicate the CPU usage when the NCP service is running and processing the received packets. (a) Jetway. (b) Lantiq. (c) Raspberry.

for the three hardware platforms. The main indication from these trials is that the NCP can manage a large number of packets, well beyond that expected in realistic scenarios.

3) Activation of Behavioral Rules: We measured the NCP latency to start to cover for a device after the notification of power state transitions. In this experiment, we initially registered a given set of rules for a large number of devices (1 PING, 1 WoC, 1 TCP KeepAlive, 1 HrtBt); then, we put to standby one device at a time, and we measured the latency at each step.

There are two different factors contributing to the overall latency: the time taken to activate the behavioral rule (i.e., to carry out any preliminary task for the specific action), shown in Fig. 8(a), and the time to set up the filtering engine (i.e., to instruct the Pcap library to pick up specific packets intended to covered devices), shown in Fig. 8(b).

We clearly see that the setup of the filtering engine is the most critical issue, as it raises up to many seconds as the number of devices increases; platforms with small processing capability are mostly affected. As shown in Fig. 8(b), the latency has a sharp exponential increase with more than 100 devices. The poor behavior is ascribable to the Pcap library: Berkeley Packet Filters are not conceived neither for handling very long filtering strings (as what happens with many devices and/or many behavioral rules) nor for frequently changing the filter parameters (as what happens every time a device suspends or resumes).

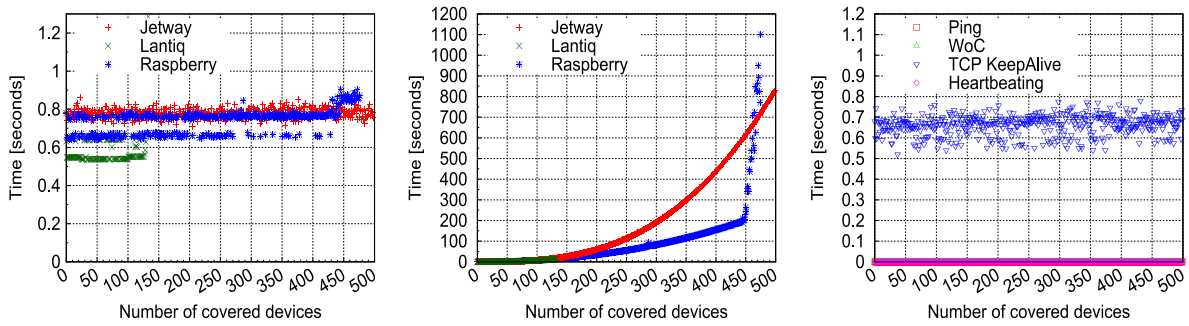


Fig. 8. NCP latency to start to cover for a device, for an increasing number of already-covered devices. Every device registered 1 PING, 1 WoC, 1 TCP KeepAlive, and 1 Heartbeating. (a) Activation of rules. (b) Setup of the filtering engine. (c) Breakdown of latency for each rule.

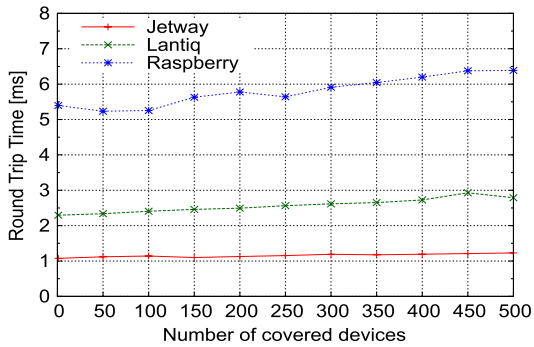


Fig. 9. Average RTT for ICMP echo-request packets addressed to sleeping hosts.

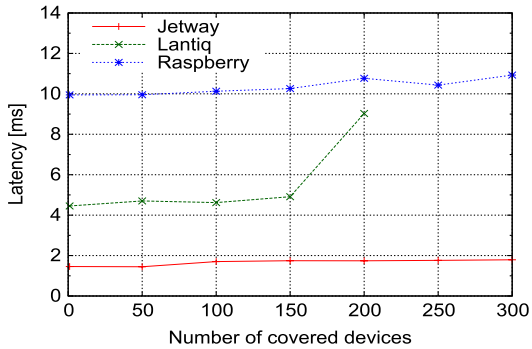


Fig. 10. Latency measured between the TCP SYN segment of an incoming connection addressed to a sleeping host and the WoL packet sent by the NCP. The curve for Lantiq ends at 200 devices because of the limited memory available on that platform.

Latencies larger than a few seconds are not compatible with NCP operation, because packets intended to sleeping hosts are lost in this interval. The graph in Fig. 8(b) can be used to assess the maximum number of clients the NCP is able to cover simultaneously; by accepting a delay of up to 1 s, this number is approximately 40–50, depending on the platform.

Coming back to the activation latency, the breakdown in Fig. 8(c) shows that most of the time is taken by the TCP KeepAlive rule. This kind of rule infers TCP sequence numbers at activation; however, in our emulation, we did not really have active TCP sessions, and thus, a timeout of 500 ms expires before giving up. In real situations, TCP-KA takes the same time as other rules (a few milliseconds).

4) NCP Operation: We are interested in evaluating how the presence of the NCP impacts the rest of the network. We expect the performance to get worse when a larger number of devices are involved; thus, we again considered how our implementation scales with an increasing number of covered devices. We carried out our analysis for PING, WoC, and Heartbeating rules.

For PING and WoC, we evaluated the latency seen by remote hosts when the NCP processes packets. We registered an increasing number of client devices, put all of them in standby, and generated a large number of ICMP echo requests and TCP SYN packets addressed to one of the sleeping devices.

Fig. 9 shows the average round-trip time (RTT) for ICMP echo requests. It slightly increases with the number of covered devices. Furthermore, we can argue that the RTT is about one order of magnitude larger than typical values measured in LANs (a few hundred microseconds). This time, too, the reason is ascribable to the Pcap library, which returns packets to the processing engine after a short timeout.

For WoC, we measured how much time the NCP takes to send out the WoL packet after an incoming connection is seen. Fig. 10 shows that the latency slightly increases with the number of covered devices. In this case, we argue that the delay experienced by the user is indeed the sum of the latency shown in Fig. 10 plus the time the device takes to resume to full operation. The latter does not depend on NCP operation, but only on the device's hardware and OS. We verified that the overall latency is well tolerated by a human that is trying to connect to a sleeping host (see Section V-B6).

For what concerns Heartbeating, assuming that the typical time intervals are within the range of hundreds of seconds or dozens of minutes, performance does not matter. However, heartbeating over TCP requires transparent and seamless migration of the session between the client device and the NCP; this operation must be complete before the client falls into standby. Hence, we measured the time taken to suspend the TCP session and to transfer its current state to the NCP (see Table III). Latency includes the time to freeze the TCP session, to save its status, and to transfer the status to the NCP through the UPnP interface. The time to resume the session is not taken into account because the session is only restored after the notification that the host is entering the standby state. Nevertheless, the latency to resume the session is only a few milliseconds [21].

TABLE III
PERFORMANCE ANALYSIS FOR TCP MIGRATION

	Min	Average	Max	Std. dev.
1 NCP client registered	128 ms	131 ms	133 ms	1.58 ms
10 NCP clients registered	128 ms	132 ms	136 ms	0.24 ms

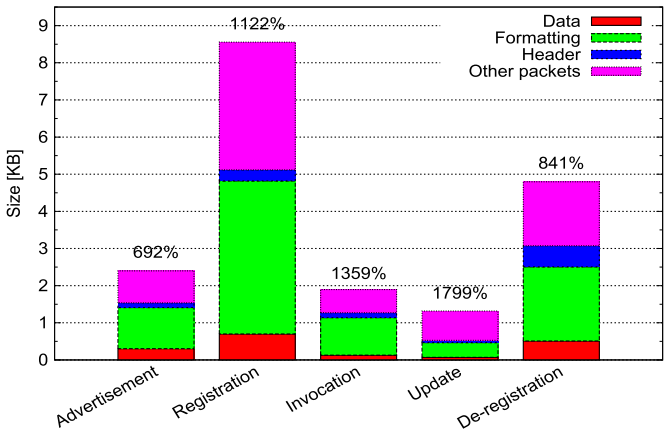


Fig. 11. Traffic overhead brought by the UPnP protocol for advertisement, registration of new devices, invocation of behavioral actions, update of state variables, and deregistration of devices.

Figures shown in Table III are small; however, they must be related to the mechanism implemented to go to standby. Indeed, if applications are simply notified once this process is initiated, such values are not sufficient to guarantee that the procedure completes before the network interface and the host are shut down. Instead, if applications could slow down such process, there would be no problems.

5) Communication Overhead: Latencies entailed by UPnP are implicitly included in the analysis in Section V-B1 and amount to few dozen milliseconds. One major drawback of UPnP is the large amount of overhead introduced to transmit even few bytes of data. In our case, actual data are parameters such as device identification, IP and MAC addresses, device description, service identifiers, and state variables; this information is formatted in XML and encoded. Then, UPnP-specific and other network (e.g., TCP and IP) headers are added to build the packet. Finally, additional information may be exchanged to implement the UPnP semantics. Hence, we evaluated the efficiency of UPnP communication by breaking down the whole bunch of data transmitted over the network into the following terms: real data/information carried by packets, XML formatting, UPnP and other network headers, and overhead due to additional semantic packets. Results are summarized in Fig. 11. The overhead of the registration phase is quite large due to the retrieval of UPnP device and service descriptions. However, the overhead is small during the notification of state variable changes, which corresponds to power state transitions.

6) Transition Times: Transition between idle and standby is a critical factor for two reasons. First, optimal strategies to put devices in standby are rather difficult to design, given the large number of short idle intervals and the substantial unpredictability of the network requests [2]. Second, the transition from standby to active affects the responsiveness of the device; despite the hardware improvement, this transition still takes up to a few seconds. These considerations raise the worry that frequent power state transitions may result in larger power consumption than the idle state, in addition to all concerns about the latency to return to normal operation.

TABLE IV
SWITCHING TIMES AND ESTIMATION OF THE MINIMAL STANDBY PERIOD FOR SOME LAPTOPS

Model	OS	$T_{i2s}[s]$	$T_{s2i}[s]$	$T_{s,min}[s]$
Dell Inspiron 910	Windows XP	15.1	10.5	1.71
Dell Latitude D531	Xubuntu 11.04	5.5	5.0	0
Dell Latitude D531	Windows 7	12.0	5.5	0
Toshiba Satellite A205	Ubuntu 11.10	7.5	4.6	1.65
Toshiba Satellite A205	Windows 7	4.5	8.0	2.40

TABLE V
ESTIMATED STANDBY TIME UNDER DIFFERENT NCP BEHAVIORS FOR OFFICE (FIRST FIGURE) AND HOME (SECOND FIGURE) ENVIRONMENTS

	B-1	B-2	B-3	B-4
Sleep (% idle time)	20–53	48–76	90–99	87–97
Sleep (% absolute)	12–31.8	28.8–45.6	54–59.4	52.2–58.1

We measured the energy and the time taken for state transitions (from idle to standby and vice versa) on some computers available in our laboratory. We also computed the minimum standby period to achieve a positive energy balance for the whole cycle. A positive balance means that the energy consumed to enter and to exit the standby is no more than the energy that would have been spent in idle for the same period of time.

Table IV reports the outcomes for different devices. T_{i2s} and T_{s2i} are the average transition times from idle to standby and vice versa, respectively; $T_{s,\min}$ is the minimum sleeping time to achieve a positive energy balance. We note that very short intervals are necessary to guarantee a positive energy balance, despite the long times taken by the hardware under examination to enter/exit standby; hence, it is almost always convenient to switch to standby.

Although our estimation is far to be accurate and complete, it gives a clear indication about the order of magnitude of the variables we are seeking for. The results confirm the convenience to switch to standby even for very short periods of time.

C. Expected Energy Savings

The usage of the NCP saves energy, because covered devices draw less power for the time they spend in low-power states. However, the NCP consumes energy as well, and this must be taken into account as a pejorative factor. We can write the expression for the overall power saving as

$$P_{\text{Saving}} = N(P_{\text{Idle}} - P_{\text{Sleep}})f_{\text{Sleep}} - P_{\text{NCP}} \qquad (1)$$

where P_{Saving} is the overall estimation for the power that could be saved, P_{Idle} is the average power consumption in idle state,

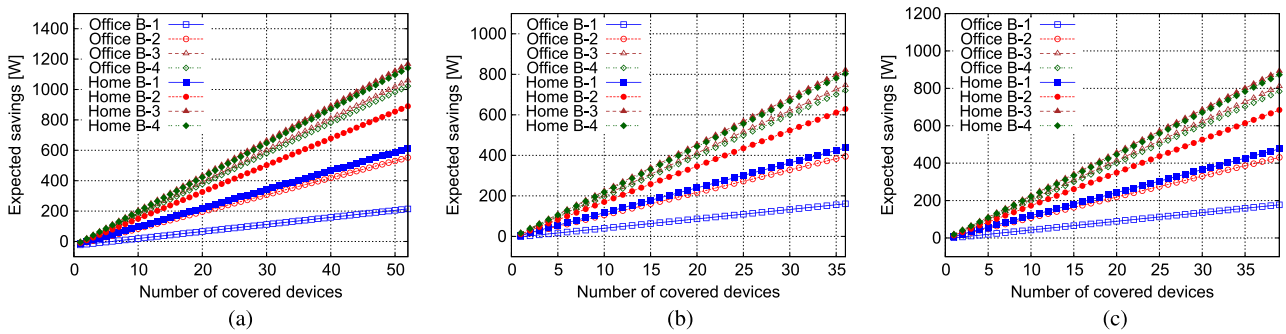


Fig. 12. Expected energy savings for a typical desktop PC in home and office environments, considering different sets of capabilities for the NCP. (a) Jetway. (b) Lantiq. (c) Raspberry.

P_{Sleep} is the average power consumption in standby mode, N is the total number of devices covered by the NCP service (its maximum value depends on the hardware hosting the NCP service), P_{NCP} is the power drawn by the device hosting the NCP (see Table II), and f_{Sleep} is the fraction of time the devices can remain in low-power mode.

Meaningful estimation of energy saving relies on how long the devices can sleep, which mainly depends on their usage pattern, the set of covered applications, and the network traffic profile. To simplify the analysis, we consider four kinds of behaviors in two representative environments [1]. The behaviors correspond to plausible capabilities for an NCP: 1) B-1: wake up the device on any packet, except ignorable traffic (HSRP, PIM, OSPF, etc); 2) B-2: wake up the device on any packet, except with those requiring simple mechanical response (ARP, ICMP, IGMP, etc.); 3) B-3: proxy a small set of applications such as telnet, ssh, vnc, file sharing, and NetBIOS; and 4) B-4: proxy all kinds of traffic, except scheduled tasks, such as regular network backups, antivirus, or software updates. The current stage of our implementation roughly corresponds to B-3, which was shown to be one of the most effective solutions [1]. Representative environments for application of the NCP are homes and offices.

We use previous surveys on usage patterns to estimate the fraction of time that devices can spend in standby [1]. Table V shows such figures, assuming that devices are idle 60% of the time (a reasonable value inferred by figures shown in those surveys) and a transition time[5] of 10 s (rather conservative for modern devices). We assume 41 and 2.3 W as average idle and standby power consumption, respectively.[6] For the hardware platforms hosting the NCP service, we consider full load power consumption as a worst case analysis (see Table II). It is worth noting that, in case of network equipment (such as the home gateway), the power consumption of such devices might not be considered, as these devices would be anyway active, even without the presence of the NCP.

The setup of the filtering engine limits the number of devices that can be covered simultaneously by the NCP, as shown in Fig. 8(b); we take into account this constraint in the computation of the expected energy savings. Considering 1-s delay as a likely threshold to avoid any packet loss, up to 52, 36, and 39 devices can be covered simultaneously by the Jetway, Lantiq, and Raspberry platforms, respectively.

Fig. 12 shows the expected savings under the aforementioned hypotheses. It rises up quickly to several hundred watts for ten devices. We also observe the large difference between B-3 and B-1, which justifies the interest in our NCP architecture. Finally, we remark that the power budget may be negative for a small number of devices, even if this cannot be appreciated immediately by the scale in Fig. 12. In particular, there must be at least two to five devices when the Jetway platform is used, depending on the specific environment and behavior considered; that is clearly due to its larger power consumption. In addition, the Lantiq home gateway must cover for at least two devices in the B-1/office scenario; however, one single covered device provides a positive energy budget in all other situations.

VI. Conclusion

We have discussed an architecture to realize the concept of NCP and its software implementation. Our work has focused on a set of generalized behavioral rules, rather than tackling specific applications and protocols. We have extended previous work on this topic by investigating practical and performance matters in implementing the NCP and by considering the energy budget in realistic scenarios.

Our work has pointed out some scalability concerns that limit the number of devices that can be covered simultaneously by the NCP. However, we have also shown that the NCP could be exploited with great energy saving in practice; its architecture is thin enough to run on low-power and embedded devices, and it leads to positive energy budgets even with few devices. Hardware platforms to run this software must be chosen according to the number of devices to be covered. The possibility to run the NCP on home gateways makes its application convenient even with only a single host (such as in home scenarios).

We plan to carry on our work by a more thorough performance evaluation for each single behavioral rule, looking for additional scalability constraints. Furthermore, we also aim at conducting additional estimations of the energy budget under

[5]The transition time is defined as the shortest time the device remains active after a wake up [1]. It consists of the time to switch from standby to active and vice versa.

[6]Values taken from Energy Star office equipment calculator: http://www.eu-energystar.org/en/en_008.shtml.

different conditions (e.g., by considering other usage and traffic profiles) and at developing analytic models to this purpose. Finally, it would be interesting to have some form of prediction for incoming traffic, in order to reduce the wake-up latency for delay-sensitive applications.

REFERENCES

[1] S. Nedevschi *et al.*, "Skilled in the art of being idle: Reducing energy waste in networked systems," in *Proc. 6th USENIX Symp. NSDI*, Boston, MA, USA, Apr. 22–24, 2009, pp. 381–394.

[2] K. Christensen, P. Gunaratne, B. Nordman, and A. George, "The next frontier for communications networks: Power management," *Comput. Commun.*, vol. 27, no. 18, pp. 1758–1770, Dec. 2004.

[3] C. Gunaratne, K. Christensen, and B. Nordman, "Managing energy consumption costs in desktop PCs and LAN switches with proxying, split TCP connections, and scaling of link speed," *Int. J. Netw. Manage.*, vol. 15, no. 5, pp. 297–310, Sep./Oct. 2005.

[4] "Advanced Configuration and Power Interface Specification," Revision 5.0, Dec. 2011.

[5] "Energy Start Program Requirements Product Specification for Computers," Version 6.0, Oct. 2013.

[6] M. Jimeno, K. Christensen, and B. Nordman, "A network connection proxy to enable hosts to sleep and save energy," in *Proc. IEEE IPCCC*, Austin, TX, USA, Dec. 7–9, 2008, pp. 101–110.

[7] K. Christensen and F. Gulledge, "Enabling power management for network-attached computers," *Int. J. Netw. Manage.*, vol. 8, no. 2, pp. 120–130, Mar./Apr. 1998.

[8] R. Bolla, M. Giribaldi, R. Khan, and M. Repetto, "Design and implementation of cooperative network connectivity proxy using universal plug and play," in *The Future Internet — Future Internet Assembly 2013: Validated Results and New Horizons*, vol. 7858, ser. Lecture Notes in Computer Science, A. Galis, and A. Gavras, Eds. New York, NY, USA: Springer-Verlag, May 2013, pp. 323–335.

[9] R. Bolla, M. Giribaldi, R. Khan, and M. Repetto, "Network connectivity proxy: An optimal strategy for reducing energy waste in network edge devices," in *Proc. 24th TIWDC Green ICT*, Sep. 23–25, 2013, pp. 1–6.

[10] R. Bolla, M. Giribaldi, R. Khan, and M. Repetto, "Design of home energy gateway boosting the development of smart grid applications at home," in *Proc. 4th ICEAC*, Istanbul, Turkey, Dec. 16–18, 2013, pp. 103–108.

[11] L. Irish and K. Christensen, "A 'green TCP/IP' to reduce electricity consumed by computers," in *Proc. IEEE Southeastcon*, Orlando, FL, USA, Apr. 24–26, 1998, pp. 302–305.

[12] J. Klamra, M. Olsson, K. Christensen, and B. Nordman, "Design and implementation of a power management proxy for universal plug and play," in *Proc. SNCNW*, Halmstad, Sweden, Nov. 23–24, 2005.

[13] M. Jimeno and K. Christensen, "A prototype power management proxy for Gnutella peer-to-peer file sharing," in *Proc. IEEE Conf. Local Comput. Netw.*, Dublin, Ireland, Oct. 15–18, 2007, pp. 210–212.

[14] P. Werstein and W. Vossen, "A low-power proxy to allow unattended Jabber clients to sleep," in *Proc. 9th Int. Conf. PDCAT*, Dunedin, New Zealand, Dec. 1–4, 2008, pp. 390–395.

[15] Y. Agarwal *et al.*, "Somniloquy: Augmenting network interfaces to reduce PC energy usage," in *Proc. 6th USENIX Symp. NSDI*, Boston, MA, USA, Apr. 22–24, 2009, pp. 365–380.

[16] Y. Agarwal, S. Savage, and R. Gupta, "SleepServer: A software-only approach for reducing the energy consumption of PCs within enterprise environments," in *Proc. USENIX ATC*, Boston, MA, USA, Jun. 23–25, 2010, pp. 22:1–22:15.

[17] R. Bolla, R. Bruschi, M. Giribaldi, R. Khan, and M. Repetto, "Smart proxying for reducing network energy consumption," in *Proc. Int. SPECTS*, Genoa, Italy, Jul. 8–11, 2012, pp. 1–8.

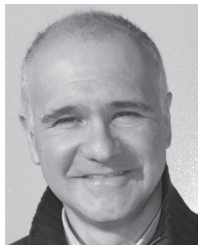
[18] Magic Packet Technology, Whitepaper, AMD, Sunnyvale, CA, USA, Nov. 1995.

[19] D. C. Plummer, "An Ethernet Address Resolution Protocol–Or–Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware," RFC 826, Nov. 1928.

[20] S. Cheshire, "IPv4 Address Conflict Detection," RFC 5227, Jul. 2008.

[21] R. Bolla, M. Chiappero, R. Rapuzzi, and M. Repetto, "Seamless and transparent migration for TCP sessions," in *Proc. IEEE 25th Int. Symp. PIMRC*, Washington, DC, USA, Sep. 2–5, 2014.

[22] "UPnP Device Architecture," Version 1.0, Oct. 2008.



Raffaele Bolla (S'89–M'95) received the Ph.D. degree in telecommunications from the University of Genoa, Genoa, Italy, in 1994.

He is currently a Full Professor with the Department of Electrical, Electronic and Telecommunications Engineering and Naval Architecture (DITEN), University of Genoa, where he is also leading the Telecommunication Networks and Telematics Laboratory. He has been a Principal Investigator of many important research projects and contracts with both public institutions and private companies. He is also involved in many standardization activities in the ITU Telecommunication Standardization Sector, the European Telecommunications Standardization Institute (ETSI), and the IEEE (he is currently the rapporteur of a Working Item of the ETSI EE-TC). He has coauthored over 200 scientific publications in international journals and international conference proceedings. His current research interests are in networking architectures, protocols, and techniques for the smart energy grid; mechanisms and techniques for energy consumption reduction in Internet Protocol networks; advance platforms for future Internet nodes (flexible software router); software-defined network and network function virtualization based architectures; and advance management of user mobility in packet networks.

Maurizio Giribaldi received the master degree in electronic engineering from the University of Genoa, Genoa, Italy, in 1988.

In 1989, he joined Marconi SpA (now part of the Ericsson group), where he worked as a Hardware and Embedded Real-Time Software Designer. In 2000, he took the responsibility of the System Engineering team of the Marconi Optic-Multi-Service Product division. In 2004, he joined the Optical Multi-Service Networks Product Group of Alcatel-Lucent Italia SpA as a System Architect Manager. Since 2012, he has been a Technical Director with Infocom, Genoa, where he is responsible for the company's R&D activities.

Rafiullah Khan was born in Kohat, Pakistan, in 1987. He received the B.Sc. degree in electrical engineering from NWFP University of Engineering and Technology, Peshawar, Pakistan, in 2009, the Master's degree in satellite navigation and related applications from Politecnico Di Torino, Turin, Italy, in 2010, and the Ph.D. degree from the University of Genoa, Genoa, Italy, in 2015, under the European Commission's funded Erasmus Mundus program.

He has coauthored several papers in international conference proceedings. His Ph.D. research area was green networking, with particular focus on energy-aware home networking. His research interests include ad hoc networking, self-organizing networks, and green networking.

Matteo Repetto received the Ph.D. degree in electronics and computer science from the University of Genoa, Genoa, Italy, in 2004.

From 2004 to 2009, he was a Postdoctoral Fellow with the University of Genoa. Since 2010, he has been a Research Associate with the National Inter-University Consortium for Telecommunications (CNIT), Genoa. He has been teaching many courses in telecommunication networks and network security. He has been involved in many different national and European research projects in the networking area. He has coauthored over 20 scientific publications in international journals and conference proceedings. His current research interests include wireless networks, estimation of freeway vehicular traffic, pervasive communications and mobility management, and energy-efficient networking.